Notes about possible technical criteria for evaluating institutional repository (IR) software

Andy Powell

UKOLN, University of Bath

December 2005

Introduction

This document attempts to identify some of the technical criteria that might be used to evaluate the different institutional repository (IR) software platform options, particularly in terms of the 'machine' interfaces that the repository offers. The list of issues is not intended to be exhaustive, and the approach is based on the assumption that other, non-technical, criteria such as usability and configurabilty have already received detailed consideration in other documents.

Brief overview of main IR software

Three of the most popular IR software platforms are DSpace[1], ePrints.org[2] and Fedora[3] (though there are others of course). Trying to compare these three is a little like comparing apples and oranges. DSpace is a Java-servlet application that runs under Apache Tomcat. EPrints.org is written in Perl and typically runs under Apache, using mod-perl to improve performance. Both applications provide the basis for an IR 'out of the box', including an end-user Web interface and so on. Both offer similar functionality to the end-user. Fedora on the other hand is more like a software toolkit. It provides the underlying IR framework, but requires custom development of a user-interface, either by layering an existing suite of user-interface tools on top of the Fedora APIs, or by building from scratch.

Any decision about which IR software platform to choose must be based not only on the technical and functional capabilities of the system but also in determining best fit with organisational IT strategy and with the availability of local software development effort. However, as a way of helping with that decision making process, it may be sensible to ask the developers of these software platforms to respond to the issues raised in the sections below. Some potential questions are suggested in each section.

Finally, it is perhaps also worth noting that the author has little or no recent experience of installing any of these software platforms. Therefore, no attempt has been made to comment here on the specific capabilities of the three systems with respect to the issues below. Detailed experience of installing, configuring, using and preferably developing against each of them would be a significant advantage in

[1] DSpace
< http://www.dspace.org/>

[2] EPrints
< http://www.eprints.org/>

[3] fedora
< http://www.fedora.info/>

being able to provide a proper comparison of their technical strengths and weaknesses.

Technical criteria

The outcomes of the ePrints UK project[4] and, more recently, discussions with colleagues at SURF have raised a number of technical issues that need to be addressed in the area of IRs. These are described in more detail below.

Complex objects

Repositories, particularly eprint archives, have tended to be developed around relatively simple 'single item' objects. Even where the IR handles multiple versions and/or formats of the same item, there tends to be a single metadata record for the item, linking to the multiple versions/formats.

In the case of learning object repositories (LOR) it is well understood that much of the content that will be deposited will be in the form of IMS Content Packages (i.e. reasonably tightly-coupled complex bundles of resources). The same is also likely to be true of eprint archives and research data repositories in the fullness of time, where we are likely to see a move towards some form of packaging of 'complex objects'. Consider, for example, a typical eprint (if such a thing exists). Conceptually, an eprint consists of a 'work' and one or more manifestations of that work (a PDF file, a Word document, etc.). Each of these things may also have separate metadata records associated with them. Being able to bundle these separate chunks of content and metadata together in some form, wrapped in a METS or MPEG-21 DID package, will simplify (at least in the long term) the way that these kinds of objects can be deposited, managed and retrieved from repositories.

In order that 'complex objects' can be dealt with in a fully automated and interoperable way we need to develop complex object models (i.e. an agreed way to model the works and manifestations described above). We also need agreed mechanisms for instantiating those models in concrete syntaxes such as XML.

Although building support for one or more of the current packaging standards into repository software should be relatively straight-forward, software may also need to have some knowledge about the 'complex object models' being used. Without this knowledge, IR software will be able to unbundle a package into its component parts, but it will not understand the relationships between the component parts in order that actions can be performed on them in sensible ways.

In the general case, the issues associated with sharing knowledge about the modelling constructs being used within complex objects are non-trivial. The author suggests that this is a 'semantic Web' issue that requires significant research work. In specific cases, it may be possible to agree particular 'complex object' models for particular applications (a model for eprints, a model for datasets, a model for lecture objects, etc.). But even if this approach is taken, designers of IR software will need to marry their potentially complex internal data-structures with the externally visible packaging standards accordingly to each of the chosen models, as data flows in and

[4] The ePrints UK Project – Final report
< http://www.rdn.ac.uk/projects/eprints-uk/docs/final-report/eprints-uk-final-20050316.pdf>

out through the repository APIs (search, harvest, deposit, delete, obtain). It is not clear how easy it will be to do this in an open-ended and flexible way.

The kinds of questions that could be put to IR software developers include:

- Does your software provide any support for 'complex objects' (i.e. packages)?
- If so, what packaging standards does it support (IMS-CP, METS, MPEG-21 DID, other)?
- How are 'complex objects' handled within the IR? Are they treated as a single blob of data, or does the IR unbundled them into component parts in some way?
- How are 'complex objects' handled by the interfaces (services) offered by the IR (search, harvest, deposit, delete, obtain)?
- Have any attempts been made to document (and share) the 'complex object models' used within the IR?
- If the system allows 'complex objects' to be unbundled into their component parts, can the components be re-assembled into a complex object? Can component parts from multiple 'complex objects' be combined into a new 'complex object'?

Metadata

It is clear that the use of simple Dublin Core metadata and the rather loosely coupled bundles of related objects found in many IRs (as described above) leads to problems for the consumers of metadata from those systems. As the ePrints UK project found, it is difficult or impossible in many cases to reliably tie identifiers and metadata records to individual 'manifestations' of eprints (i.e. different formats), largely because of the widely varying practices across institutions. In practice this means that it is often difficult for software robots to move reliably from the harvested metadata record to the full-text of an eprint.

Moving to richer metadata, for example as offered by qualified Dublin Core, may help with this problem but only partially. The fundamental problem lies in our lack of an agreed way to model, instantiate and handle complex objects (see above). However, it is probably the case that moving to the use of qualified Dublin Core metadata in the current 'simple object' environment would improve things. For example, it would be very useful to agree a set of guidelines for how to use qualified DC to describe eprints (along the same lines as the current guidelines for using simple DC to describe eprints[5]), possibly based on the DC Libraries Application Profile.

The kinds of questions that could be put to IR software developers include:

- Do you offer support for qualified Dublin Core?

[5] Using simple Dublin Core to describe eprints <http://www.rdn.ac.uk/projects/eprints-uk/docs/simpledc-guidelines/>

- If so, how configurable is this aspect of the IR (i.e. does it support arbitrary application profiles - sets of metadata terms, qualifiers and controlled vocabularies)?
- Is qualified DC also supported through the interfaces (services) offered by the IR (search, harvest, deposit, delete, obtain)?

Identifiers

IR software should be able to uniquely identify all the objects of interest stored within, and associated with, the repository. Note that this may include digital objects (files, metadata records, etc.), physical objects (authors, books, etc.) and conceptual objects (vocabulary terms, works, etc.). For each of these identifiers, the IR software should be able to serve a reasonable representation of the identified object.

Again, this issue is very closely related to the 'complex object' modelling issue above. Some IR systems may be limited in this respect. For example, if 'complex objects' are treated as blobs of data within the system, then it may not be possible to identify (and hence obtain a representation of) a particular component within the 'complex object'. If identifiers are assigned to an eprint 'work', then it may not be possible to uniquely identify (and hence obtain a representation of) a particular 'manifestation' of that work.

The kinds of questions that could be put to IR software developers include:

- How are identifiers assigned within the IR?
- Are all objects within the IR assigned a unique and persistent identifier? This includes conceptual works, document manifestations, authors, contributors and other 'agents', metadata records, 'complex object' packages, terms in vocabularies and so on). Are some objects not assigned a public identifier?
- Is it possible to 'resolve' that identifier in order to obtain a representation of the object?
- How is the persistence and uniqueness of identifiers guaranteed?
- How are object identifiers exposed through the interfaces (services) offered by the IR (search, harvest, deposit, delete, obtain)?

Machine interfaces (services)

Each IR software platform is likely to offer a range of machine interfaces (or services) to the content and functionality it offers. These interfaces need to be documented and, where possible, conform to agreed standards. As a community, we need to try and adopt a limited set of interface definitions. Several are already well known and widely adopted, including HTTP, XML, DC, IEEE LOM, Z39.50, SRW/SRU, OAI-PMH, RSS, OpenURL, METS, MPEG-21 DID, IMS-CP. Providing a machine interface for 'deposit' is currently less well standardised. Developing such an interface (assuming it were to be widely adopted) would allow third-parties to develop tools such as content packagers that could automatically deposit a newly created package into a repository. Further, such tools could work against any IR that supported the same 'deposit' interface.

Recent discussions have identified a number of candidate protocols for a 'deposit' interface including HTTP POST, SRW/SRU Update, the Fedora deposit interface, WebDAV, and the Atom Publishing Protocol. Some work needs to be undertaken to evaluate which, if any, of these is most appropriate for widespread adoption or whether there are other better candidates.

The kinds of questions that could be put to IR software developers include:

- What kinds of machine interfaces does your software support?
- What (open) standards do your interfaces conform to?
- Does your IR support a 'deposit' interface? If so, what is it?
- How is metadata and content passed across the interface? Are there any limitations in what can be carried?